```
================================================================================
Demonstration of the “Obelisk” proof assistant.
Variant: higher-order logic
 Author: Georgy Dunaev
  Email: contact@obelisk.mozmail.com
   Date: Fri Aug 07 11:39:55 2026
License: All rights reserved.©
================================================================================
```

1) Propositional logic:

1.01) Axm. A01:

$$(⟦\alpha⟧\to(⟦\beta⟧\to⟦\alpha⟧))$$

1.02) Axm. A02:

$$((⟦\alpha⟧\to(⟦\beta⟧\to⟦\gamma⟧))\to((⟦\alpha⟧\to⟦\beta⟧)\to(⟦\alpha⟧\to⟦\gamma⟧)))$$

1.03) Axm. A03:

$$((⟦\alpha⟧\wedge⟦\beta⟧)\to⟦\alpha⟧)$$

1.04) Axm. A04:

$$((⟦\alpha⟧\wedge⟦\beta⟧)\to⟦\beta⟧)$$

1.05) Axm. A05:

$$(⟦\alpha⟧\to(⟦\beta⟧\to(⟦\alpha⟧\wedge⟦\beta⟧)))$$

1.06) Axm. A06:

$$(⟦\alpha⟧\to(⟦\alpha⟧\vee⟦\beta⟧))$$

1.07) Axm. A07:

$$(⟦\beta⟧\to(⟦\alpha⟧\vee⟦\beta⟧))$$

1.08) Axm. A08:

$$((⟦\alpha⟧\to⟦\gamma⟧)\to((⟦\beta⟧\to⟦\gamma⟧)\to((⟦\alpha⟧\vee⟦\beta⟧)\to⟦\gamma⟧)))$$

1.09) Axm. A09:

$$((\neg⟦\alpha⟧)\to(⟦\alpha⟧\to\bot))$$

1.10) Axm. A10:

$$((⟦\alpha⟧\to⟦\beta⟧)\to((⟦\alpha⟧\to\neg⟦\beta⟧)\to\neg⟦\alpha⟧))$$

1.11) Axm. A11:

$$(\bot\to⟦\alpha⟧)$$

1.12) Axm. A12:

$$(⟦\alpha⟧\vee\neg⟦\alpha⟧)$$

1.13) ***Rule*** ***Modus Ponens***:

⟦α⟧
(⟦α⟧→⟦β⟧)
---------- (MP)
⟦β⟧

2) Predicate logic:

2.01) ***Axm.*** ***Discourse domain is non-empty***:

(∃(K:Class)⊤)

2.02) ***Rule*** ***Forall Instantiation***:

⟦term t is free for substitution instead of all samely typed free occurences of the variable x in the formula φ⟧
---------- (FI)
((∀({x}:⟦X⟧)⟦φ⟧)→⟦φ[t/x]⟧)

2.03) ***Rule*** ***Exists Instantiation***:

⟦term t is free for substitution instead of all samely typed free occurences of the variable x in the formula φ⟧
---------- (EI)
(⟦φ[t/x]⟧→∃({x}:⟦X⟧)⟦φ⟧)

2.04) ***Rule*** ***Forall Abstraction***:

⟦name x does not occure free as a variable of type X in formula β⟧
(⟦β⟧→⟦α⟧)
---------- (FA)
(⟦β⟧→∀({x}:⟦X⟧)⟦α⟧)

2.05) ***Rule*** ***Exists Abstraction***:

⟦name x does not occure free as a variable of type X in formula β⟧
(⟦α⟧→⟦β⟧)
---------- (EA)
((∃({x}:⟦X⟧)⟦α⟧)→⟦β⟧)

2.06) ***Rule*** ***Generalisation (admissible rule)***:

⟦α⟧
---------- (Gen)
(∀({x}:⟦X⟧)⟦α⟧)

3) Set/class-theoretic foundations of mathematics:

3.01) ***Rule*** ***Class comprehension***:

⟦(x:Name)⟧
⟦(φ:Fm)⟧
---------- (CCR)
(({x}∈{{x}|⟦φ⟧})↔(({x} set)∧⟦φ⟧))

3.02) ***Def.*** ***"Is a set" predicate***:

(∀(X:Class)((X set)↔(∃(Y:Class)(X∈Y))))

3.03) ***Def.*** ***"Is a proper class" predicate***:

(∀(K:Class)((K proper)↔(¬(K set))))

3.04) ***Def.*** ***Empty set***:

(∅={x|⊥})

3.05) ***Def.*** ***Universal class***:

(**V**={x|⊤})

3.06) ***Def.*** ***Subclass relation***:

(∀(X:Class)∀(Y:Class)((X⊆Y)↔(∀(a:Class)((a∈X)→(a∈Y)))))

3.07) ***Thm.*** ***Example: "The empty class is a subclass of any class" statement***:

(∀(X:Class)(∅⊆X))

3.08) ***Axm.*** ***"Membership respects equality from the left"***:

(∀(X:Class)∀(Y:Class)((X=Y)→∀(z:Class)((z∈X)↔(z∈Y))))

3.09) ***Axm.*** ***"Membership respects equality from the right"***:

(∀(X:Class)∀(Y:Class)((X=Y)→∀(W:Class)((X∈W)↔(Y∈W))))

3.10) ***Axm.*** ***"Extensionality"***:

(∀(X:Class)∀(Y:Class)((∀(z:Class)((z∈X)↔(z∈Y)))→(X=Y)))

3.11) ***Thm.*** ***"Intensionality"(admissible)***:

(∀(X:Class)∀(Y:Class)(((X set)∧(Y set))→((∀(W:Class)
((X∈W)↔(Y∈W)))→(X=Y))))

3.12) ***Axm.*** ***"Heredity"(a subclass of a set is also a set)***:

(∀(A:Class)∀(B:Class)(((A⊆B)∧(B set))→(A set)))

3.13) ***Axm.*** ***"Exponent"(powerclass of a set is also a set)***:

(∀(X:Class)((X set)→((X) set)))

3.14) ***Axm.*** ***"Regularity"(only universal class is equal to its powerclass)***:

(∀(X:Class)(((X)⊆X)→(**V**⊆X)))

3.15) ***Def.*** ***Transitivity predicate***:

(∀(A:Class)(Tr(A)↔(∀(B:Class)((B∈A)→(B⊆A)))))

3.16) ***Def.*** ***Ordinality predicate***:

(∀(A:Class)(Ord(A)↔(Tr(A)∧∀(B:Class)((B∈A)→Tr(B)))))

3.17) ***Def.*** ***Class of all ordinal sets(ordinal numbers)***:

(**On**={w|Ord(w)})

3.18) ***Thm.*** ***The class of all ordinals is an ordinal class***:

Ord(**On**)

3.19) ***Def.*** ***Union of the elements of a class***:

(∀(K:Class)(⋃K={x|∃(y:Class)((y∈K)∧(x∈y))}))

3.20) ***Def.*** ***Intersection of the elements of a class***:

(∀(K:Class)(⋂K={x|∀(y:Class)((y∈K)→(x∈y))}))

3.21) ***Def.*** ***Domain (includes arguments with proper class values)***:

(∀(F:Class)(**Dom**(F)={x|∃(b:Class)((x,b)∈F)}))

3.22) ***Def.*** ***IF-THEN-ELSE-FI(meta,semantic,Class-specific,WIP, TODO: add side condition that v is not free in psi, add (W,G|-<... = ... >)***:

(∀(v:Variable)∀(ψ:Fm)∀(A:Class)∀(B:Class)((IF ⟦ψ⟧ THEN A ELSE B FI)={⦃v⦄|((⟦ψ⟧∧(⦃v⦄∈A))∨((¬⟦ψ⟧)∧(⦃v⦄∈B))}))

3.23) ***Def.*** ***if-then-else-fi(meta,syntactic,polymorphic over one type,non-dependent,TBI)***:

(ToDo: “simple_if”, a non-dependent type-theoretic verison of the conditional, which works not only for Class, but also for any other type)

3.24) ***Def.*** ***If-Then-Else-Fi(meta,syntactic,polymorphic over one value,dependent,TBI)***:

(ToDo: Add “typeof(if b then t1 else t2) := [| simple_if b then typeof(t1) else typeof(t2) fi |]”; simple_if is an instance using polymorphic over one metatype variable; The expression inside [|-|] brackets has type Type.)

3.25) ***Def.*** ***Evaluation of a [function|class-function|family]***:

(∀(F:Class)∀(a:Class)(F(a)=(IF (∃(s:Class)((a,s)∈F)) THEN ⋃{s|(a,s)∈F} ELSE **V** FI)))

3.26) ***Def.*** ***Image class***:

(∀(f:Class)∀(A:Class)(f[A]={b|∃(a:Class)((a∈A)∧(f(a)=b))}))

3.27) ***Def.*** ***“Has exactly one element” predicate***:

(∀(s:Class)((s singleton)↔(∃!(x:Class)(x∈s))))

3.28) ***Def.*** ***“Is set-valued” predicate***:

(∀(F:Class)((F set-valued)↔(∀(x:Class)((x∈**Dom**(F))→(F(x) set)))))

3.29) ***Def.*** ***"Has concentrated set values" predicate***:

(∀(F:Class)((F concentrated set values)↔(∀(x:Class)((x∈**Dom**(F))→((F(x) set)→∃!(y:Class)((x,y)∈F))))))

3.30) ***Def.*** ***"Has flat proper values" predicate***:

(∀(F:Class)((F flat proper values)↔(∀(x:Class)((x∈**Dom**(F))→((F(x) proper)→∀(b:Class)(((x,b)∈F)→(b singleton)))))))

3.31) ***Def.*** ***"Is a family" predicate***:

(∀(F:Class)((F family)↔((F concentrated set values)∧(F flat proper values))))

3.32) ***Def.*** ***"Is a class-function" predicate***:

(∀(F:Class)((F class-function)↔((F family)∧(F set-valued))))

3.33) ***Def.*** ***"Is injective" predicate***:

(∀(F:Class)((F injective)↔(∀(a:Class)∀(b:Class) ((((a∈**Dom**(F))∧(b∈**Dom**(F)))∧(F(a)=F(b)))→(a=b)))))

3.34) ***Def.*** ***"Is a function" predicate***:

(∀(f:Class)((f function)↔((f class-function)∧(f set))))

3.35) ***Def.*** ***"Is not strictly larger than" binary relation***:

(∀(A:Class)∀(B:Class)((A≼B)↔(∃(f:Class)(((f class-function)∧(f[A]⊆B))∧(f injective)))))

3.36) ***Def.*** ***Class cardinality***:

(∀(K:Class)(|K|=(**On**∩⋂{a|(a∈**On**)∧(K≼a)})))

3.37) ***Def.*** ***"Is strictly smaller than" binary relation***:

(∀(A:Class)∀(B:Class)((A≺B)↔(¬(B≼A))))

3.38) ***Axm.*** ***Size limitation axiom related statement***:

(∀(X:Class)((X∈**V**)↔(|X|≺|**V**|)))

3.39) ***Def.*** ***"Is hereditary" predicate***:

(∀(A:Class)(Hereditary(A)↔(∀(B:Class)((B∈A)→∀(X:Class) ((X⊆B)→(X∈A))))))

3.40) ***Def.*** ***"Is powerful" predicate***:

(∀(U:Class)(Powerful(U)↔(∀(B:Class)((B∈U)→((B)∈U)))))

3.41) ***Def.*** ***"Is pairful" predicate***:

(∀(U:Class)(Pairful(U)↔(∀(A:Class)∀(B:Class)(((A∈U)∧(B∈U))→({A,B} ∈U)))))

3.42) ***Def.*** ***"Is tight" predicate***:

(∀(U:Class)(Tight(U)↔(∀(B:Class)((B⊆U)→((B≺U)↔(B∈U))))))

3.43) ***Def.*** ***Singleton set***:

(∀(a:Class)({a}={x|x=a}))

3.44) ***Def.*** ***Unordered pair***:

(∀(a:Class)∀(b:Class)({a,b}={x|(x=a)∨(x=b)}))

3.45) ***Def.*** ***Set-theoretic models of natural numbers(TBI)***:

```
Def.:  (0:Class) =  ∅
Def.:  (1:Class) = {∅}
Def.:  (2:Class) = {∅,{∅}}
Def.:  (3:Class) = {∅,{∅},{∅,{∅}}}
Def.:  ...
```

3.46) ***Def.*** ***General definition of lists of classes which applicable not only for sets, but for proper classes too(TBI)***:

```
Def.:  mark(A,n) = { {0,n} } ∪ { ( { {{a}} , n } : Class ) | (a ∈ A)}
Def.:      [] = ∅
Def.:     [A] = mark(A,1)
Def.:   [A,B] = mark(A,1) ∪ mark(B,2)
Def.:  ...
```

3.47) ***Def.*** ***Ordered pair(Dunaev's)***:

```
Def.:   (A,B) = [A,B]
```

3.48) ***Def.*** ***Projections***:

```
Def.:  pr1(L) = { a | ({{{a}},1} ∈ L)}
Def.:  pr2(L) = { b | ({{{b}},2} ∈ L)}
Def.:  ...
```

3.49) ***Def.*** ***Ordered pair(Kuratowsky's)***:

(∀(a:Class)∀(b:Class)((a,b)={{a,a},{a,b}}))

3.50) ***Def.*** ***Binary union***:

(∀(A:Class)∀(B:Class)((A∪B)={x|(x∈A)∨(x∈B)}))

```
Special Def.:  (x ∪ y) = ⋃{x,y}                          ( bad def)
General Def.:  (A ∪ B) = { z | (z ∈ A) ∨ (z ∈ B)}        (good def)
```

3.51) ***Def.*** ***Binary intersection***:

(∀(A:Class)∀(B:Class)((A∩B)={x|(x∈A)∧(x∈B)}))

3.52) **Def.** ***"Is inductive" predicate***:

(∀(X:Class)(Ind(X)↔((∅∈X)∧∀(a:Class)((a∈X)→((a∪{a})∈X)))))

3.53) **Def.** ***The intersection class of all inductive classes is called omega***:

(∀(x:Class)((x∈ω)↔(∀(Q:Class)(Ind(Q)→(x∈Q)))))

Special Def.: ω = ⋂{ q | Ind(q) } (bad def)
General Def.: ∀ x, (x ∈ ω) ↔ (∀ Q, Ind(Q) → (x ∈ Q)) (good def)

3.54) **Axm.** ***"Infinity"(ω is a set)***:

(ω set)

3.55) **Thm.** ***Class of all sets is inductive***:

Ind(V)

3.56) **Thm.** ***Intersection of all inductive classes is an inductive class itself***:

Ind(ω)

3.57) **Thm.** ***Class of all ordinal numbers is inductive***:

Ind(On)

3.58) **Thm.** ***ω is transitive***:

Tr(ω)

3.59) **Thm.** ***ω is ordinal***:

Ord(ω)

3.60) **Thm.** ***ω is in the class of all ordinal numbers(=sets)***:

(ω∈On)

3.61) **Thm.** ***"Is cardinal" predicate***:

(∀(C:Class)(Card(C)↔(∃(K:Class)(|K|=C))))

3.62) **Thm.** ***ω is cardinal***:

Card(ω)

3.63) **Thm.** ***On is a cardinal class***:

Card(On)

3.64) **Thm.** ***Class of all sets is not a cardinal class***:

(¬Card(V))

3.65) **Def.** ***"Is a universe" predicate***:

(∀(U:Class)((U
universe)↔(((ω∈U)∧((Tr(U)∧Hereditary(U))∧(Powerful(U)∧Tight(U))))∧Pairful(U))))

3.66) Thm. *The class of all sets is a universe*:

(V universe)

3.67) Axm. *"Tarski's axiom"*:

(∀(x:Class)((x set)→∃(u:Class)(((u universe)∧(x∈u))∧(u set))))

3.68) Thm. *General form of the induction principle on a transitive class*:

(∀(K:Class)∀(A:Class)((Tr(K)∧((K∩(A))⊆A))→(K⊆A)))

3.69) Axm. *Soundness of the theory T*:

(∀(φ:Fm)((T,[]⊢φ)→eval(φ)))

3.70) Axm. *Soundness of the theory F*:

(∀(φ:Fm)((F,[]⊢φ)→eval(φ)))

3.71) Def. *Completeness of the theory F*:

(∀(φ:Fm)(eval(φ)→(F,[]⊢φ)))

3.72) Axm. *If a formula is derivable in T, then it is derivable in F*:

(∀(Γ:Ctx)∀(φ:Fm)((T,Γ⊢φ)→(F,Γ⊢φ)))

4) Metalogical laws(WIP: too much known errors in this section):

4.01) Axm. *Instantiation(meta,WIP)*:

(∀(A:Type)∀(G:Ctx.Expression)∀(φ:Fm)∀(t:eval(A).Expression)∀(v:Variable)(W,⟦G⟧⊢‹(∀(x:⟦A⟧)⟦φ⟧)→(let ⟦v⟧ = ⟦t⟧ in ⟦φ⟧)›))

4.02) Axm. *Generalisation(meta,WIP)*:

(∀(A:Type)∀(Γ:Ctx)∀(φ:Fm)∀(v:Variable)((¬(Enc(❴v❵)∈**Ctx.FV**(Γ)))→((W,Γ⊢φ)→(W,Γ⊢‹∀(❴v❵:A)⟦φ⟧›))))

(A theory W is assumed here to contain only formulas without free variables, so the variable x has no free occurences in any formula of a set of formulas [|Dec(Theory.Expression)(W)|]. (Strictly: even if the type is different from X?))

4.03) Rule *CoNecessitation(meta,WIP)*:

If a formula < F,Γ⊢φ> is derived in F from the empty context, then the formula φ is derived in F from the context Γ.

4.04) Def. *Quantifier "unique"(meta,WIP)*:

(∀(φ:Fm)((!(z:Class)⟦φ⟧)↔(∀(x:Class)∀(y:Class)(((⟦φ[(q:Class)/‹z›]⟧∧(let z = y in ⟦φ⟧))→(x=y)))))

4.05) Def. *Quantifier "exists and unique"(meta,WIP)*:

(∀(φ:Fm)(T,[]⊢‹(∃!(x:Class)⟦φ⟧)↔((∃(x:Class)⟦φ⟧)∧!(x:Class)⟦φ⟧)›))

4.06) Axm. Class comprehension axiom schema(meta,WIP):

(∀(φ:Fm)∀(x:Name)∀(k:Class.Expression)(FFI((k:Class),‹x›,(φ:Bool))→(T,[]⊢‹(⟦k⟧∈{x|⟦φ⟧})↔((⟦k⟧ set)∧(let x = ⟦k⟧ in ⟦φ⟧))›)))

5) Tactics and inference rules:

5.01) Elementary theorems:

(F,[‹B∈x›]⊢‹∃(x:Class)(B∈x)›)
(F,[‹B∈x›]⊢‹∀(B:Class)((B set)↔(∃(x:Class)(B∈x)))›)
(F,[‹B∈x›]⊢‹(B set)↔(∃(x:Class)(B∈x))›)

5.02) An example of inference rule:

(F,[‹B∈x›]⊢‹∃(x:Class)(B∈x)›)
---------- (def_to_intro th2)
(F,[‹B∈x›]⊢‹B set›)

5.03) An application of a deduction rule:

(F,[]⊢‹(B∈x)→(B set)›)

5.04) A simple theorem being proven using tactics:

(F,[‹∃(y:Class)∀(x:Class)⟦φ⟧›]⊢‹∃(y:Class)⟦φ⟧›)
(F,[‹∀(x:Class)⟦φ⟧›]⊢‹∃(y:Class)⟦φ⟧›)
(F,[‹⟦φ⟧›]⊢‹∃(y:Class)⟦φ⟧›)
(F,[‹⟦φ⟧›]⊢‹⟦φ⟧›)
Displaying the proof of that theorem:
Proof.
1 |(F,[‹⟦φ⟧›]⊢‹⟦φ⟧›)
2 |(F,[‹⟦φ⟧›]⊢‹∃(y:Class)⟦φ⟧›)
3 |(F,[‹∀(x:Class)⟦φ⟧›]⊢‹∃(y:Class)⟦φ⟧›)
4 |(F,[‹∃(y:Class)∀(x:Class)⟦φ⟧›]⊢‹∃(y:Class)⟦φ⟧›)
5 |(F,[‹∃(y:Class)∀(x:Class)⟦φ⟧›]⊢‹∀(x:Class)∃(y:Class)⟦φ⟧›)
6 |(F,[]⊢‹(∃(y:Class)∀(x:Class)⟦φ⟧)→∀(x:Class)∃(y:Class)⟦φ⟧›)
Q.E.D.

6) Incompleteness-related statements(WIP: too much known errors in this section):

6.01) Axm. Theories:

Let *W* be an arbitrary theory.
Let *Q* be Robinson's arithmetic.
Let *T* be a recursively axiomatizable theory of classes. (The base theory of the proof assistant.)
Let *F* be an extension(note: it is not recursively axiomatizable) of *T* (in the extension of the language that has all Henkin constants), which is complete and admits the reflection axiom.
The introducing of the types allows to hide both Gödel's encoding and decoding, and to keep all variables on the same metalevel.

6.02) Thm. Löb's theorem(WIP):

(∀(φ:Fm)(Q,[]⊢‹(T,[]⊢‹(T,[]⊢φ)→⟦φ⟧›)→(T,[]⊢φ)›))

6.03) ***Thm.*** ***Gödel's second incompleteness theorem(WIP)***:

(Q,[]⊢‹(¬(T,[]⊢‹⊥›))→¬(T,[]⊢‹¬(T,[]⊢‹⊥›)›)›)